Amendments to the Specification:

Please replace paragraph [0001] with the following amended paragraph:

[0001] This application claims priority to U.S. Provisional Application Serial No. 60/400,391 titled "JSM Protection," filed July 31, 2002, incorporated herein by reference. This application also claims priority to EPO Application No. 03291911.0, filed July 30, 2003 and entitled "Test And Skip Processor Instruction Having At Least One Register Operand," incorporated herein by reference. This application also may contain subject matter that may relate to the following commonly assigned co-pending applications incorporated herein by reference: "System And Method To Automatically Stack And Unstack Java Local Variables," Serial No. [[_____]]10/632,228, filed July 31, 2003~~, Attorney Docket No. TI-35422 (1962-05401)~~; "Memory Management Of Local Variables," Serial No. [[_____]]10/632,067, filed July 31, 2003~~, Attorney Docket No. TI-35423 (1962-05402)~~; "Memory Management Of Local Variables Upon A Change Of Context," Serial No. [[_____]]10/632,076, filed July 31, 2003~~, Attorney Docket No. TI-35424 (1962-05403)~~; "A Processor With A Split Stack," Serial No. [[_____]]10/632,079, filed July 31, 2003~~, Attorney Docket No. TI-35425(1962-05404)~~; "Using IMPDEP2 For System Commands Related To Java Accelerator Hardware," Serial No. [[_____]]10/632,069, filed July 31, 2003~~, Attorney Docket No. TI-35426 (1962-05405)~~; "Test With Immediate And Skip Processor Instruction," Serial No. [[_____]]10/632,214, filed July 31, 2003~~, Attorney Docket No. TI-35427 (1962-05406)~~; "Synchronizing Stack Storage," Serial No. [[_____]]10/631,422, filed July 31, 2003~~, Attorney Docket No. TI-35429 (1962-05408)~~; "Methods And Apparatuses For Managing Memory," Serial No. [[_____]]10/631,252, filed July 31, 2003~~, Attorney Docket No. TI-35430 (1962-05409)~~; "Write Back Policy For Memory," Serial No. [[_____]]10/631,185, filed July 31, 2003~~, Attorney Docket No. TI-35431 (1962-05410)~~; "Methods And Apparatuses For Managing Memory," Serial No. [[_____]]10/631,205, filed July 31, 2003~~, Attorney Docket No. TI-35432 (1962-05411)~~; "Mixed Stack-Based RISC Processor," Serial No. [[_____]]10/631,308, filed July 31, 2003~~, Attorney Docket No. TI-35433 (1962-05412)~~; "Processor That Accommodates Multiple Instruction Sets And Multiple Decode Modes," Serial No.

[[~~_____~~]]10/631,246, filed July 31, 2003, ~~Attorney Docket No. TI-35434 (1962-05413)~~; "System To Dispatch Several Instructions On Available Hardware Resources," Serial No. [[~~_____~~]]10/631,585, filed July 31, 2003, ~~Attorney Docket No. TI-35444 (1962-05414)~~; "Micro-Sequence Execution In A Processor," Serial No. [[~~_____~~]]10/632,216, filed July 31, 2003, ~~Attorney Docket No. TI-35445 (1962-05415)~~; "Program Counter Adjustment Based On The Detection Of An Instruction Prefix," Serial No. [[~~_____~~]]10/632,222, filed July 31, 2003, ~~Attorney Docket No. TI-35452 (1962-05416)~~; "Reformat Logic To Translate Between A Virtual Address And A Compressed Physical Address," Serial No. [[~~_____~~]]10/632,215, filed July 31, 2003, ~~Attorney Docket No. TI-35460 (1962-05417)~~; "Synchronization Of Processor States," Serial No. [[~~_____~~]]10/632,024, filed July 31, 2003, ~~Attorney Docket No. TI-35461 (1962-05418)~~; "Conditional Garbage Based On Monitoring To Improve Real Time Performance," Serial No. [[~~_____~~]]10/631,195, filed July 31, 2003, ~~Attorney Docket No. TI-35485 (1962-05419)~~; "Inter-Processor Control," Serial No. [[~~_____~~]]10/631,120, filed July 31, 2003, ~~Attorney Docket No. TI-35486 (1962-05420)~~; "Cache Coherency In A Multi-Processor System," Serial No. [[~~_____~~]]10/632,229, filed July 31, 2003, ~~Attorney Docket No. TI-35637 (1962-05421)~~; "Concurrent Task Execution In A Multi-Processor, Single Operating System Environment," Serial No. [[~~_____~~]]10/632,077, filed July 31, 2003, ~~Attorney Docket No. TI-35638 (1962-05422)~~; and "A Multi-Processor Computing System Having A Java Stack Machine And A RISC-Based Processor," Serial No. [[~~_____~~]]10/631,939, filed July 31, 2003, ~~Attorney Docket No. TI-35710 (1962-05423)~~.

Please replace paragraph [0016] with the following amended paragraph:

[0016] Referring now to Figure 1, a system 100 is shown in accordance with a preferred embodiment of the invention. As shown, the system includes at least two processors 102 and 104. Processor 102 is referred to for purposes of this disclosure as a Java Stack Machine ("JSM") and processor 104 may be referred to as a Main Processor Unit ("MPU"). System 100 may also include memory 106 coupled to both the JSM 102 and MPU 104 and thus accessible by both processors. At least a portion of the memory 106

may be shared by both processors meaning that both processors may access the same shared memory locations. Further, if desired, a portion of the memory 106 may be designated as private to one processor or the other. System 100 also includes a Java Virtual Machine ("JVM") 108, compiler 110, and a display 114. The JSM 102 preferably includes an interface to one or more input/output ("I/O") devices such as a keypad to permit a user to control various aspects of the system 100. In addition, data streams may be received from the I/O space into the JSM 102 to be processed by the JSM 102. Other components (not specifically shown) may include, without limitation, a battery and an analog transceiver to permit wireless communications with other devices. As noted above, while system 100 may be representative of, or adapted to, a wide variety of electronic systems, an exemplary electronic system may comprise a battery-operated, mobile cell phone such as that is shown in Figure 2. As shown, a mobile communications device includes an integrated keypad 412 and display 414. Two processors and other components may be included in electronics package 410 connected to keypad ~~410~~412, display 414, and radio frequency ("RF") circuitry 416 which may be connected to an antenna 418.

Please replace paragraph [0023] with the following amended paragraph:

[0023] Referring again to Figure 3, as noted above, the JSM 102 is adapted to process and execute instructions from at least two instruction sets. One instruction set includes stack-based operations and the second instruction set includes register-based and memory-based operations. The stack-based instruction set may include Java bytecodes. Java bytecodes pop, unless empty, data from and push data onto the micro-stack 146. The micro-stack 146 preferably comprises the top *n* entries of a larger stack that is implemented in data storage 122. Although the value of *n* may ~~be~~ vary in different embodiments, in accordance with at least some embodiments, the size *n* of the micro-stack may be the top eight entries in the larger, memory-based stack. The micro-stack 146 preferably comprises a plurality of gates in the core 120 of the JSM 102. By implementing the micro-stack 146 in gates (e.g., registers) in the core 120 of the

processor 102, access to the data contained in the micro-stack 146 is generally very fast, although any particular access speed is not a limitation on this disclosure.

Please replace paragraph [0024] with the following amended paragraph:

[0024] The second, register-based, memory-based instruction set may comprise the C-ISA instruction set introduced above. The C-ISA instruction set preferably is complementary to the Java bytecode instruction set in that the C-ISA instructions may be used to accelerate or otherwise enhance the execution of Java bytecodes. For example, the compiler 110 may scan a series of Java ~~bytes codes~~bytecodes 112 and replace one or more of such bytecodes with an optimized code segment mixing C-ISA and bytecodes and which is capable of more efficiently performing the function(s) performed by the initial group of Java bytecodes. In at least this way, Java execution may be accelerated by the JSM 102. The C-ISA instruction set includes a plurality of instructions including a test and skip instruction as mentioned above and explained below in detail.

Please replace paragraph [0026] with the following amended paragraph:

[0026] The data storage 122 generally comprises data cache ("D-cache") 124 and data random access memory ("D-RAMset") 126. Reference may be made to copending applications U.S. Serial nos. 09/591,537 filed June 9, 2000 ~~(atty docket TI-29884)~~, 09/591,656 filed June 9, 2000 ~~(atty docket TI-29960)~~, and 09/932,794 filed August 17, 2001 ~~(atty docket TI-31351)~~, all of which are incorporated herein by reference. The stack (excluding the micro-stack 146), arrays and non-critical data may be stored in the D-cache 124, while Java local variables, critical data and non-Java variables (e.g., C, C++) may be stored in D-RAM 126. The instruction storage 130 may comprise instruction RAM ("I-RAM") 132 and instruction cache ("I-cache") 134. The I-RAMset 132 may be used for "complex" micro-sequenced bytecodes or other micro-sequences or sequences of code, as will be described below. The I-cache 134 may be used to store other types of Java bytecode and mixed Java/C-ISA instructions.

Please replace paragraph [0027] with the following amended paragraph:

[0027] As explained above, the C-ISA instruction set preferably permits register-based and memory-based operations. While not the only C-ISA instruction, as noted above one such instruction is the test and skip instruction which includes at least one register operand. An exemplary embodiment of this instruction is depicted in Figure 5. The preferred embodiment of the test and skip instruction 228 comprises an opcode 230, a first register reference 232 (labeled generically as "Rd" in Figure 5), an address program bit ("P") 234, a second register reference 236 ("Rs"), W and B bits 238, and a test condition 240.

Please replace paragraph [0028] with the following amended paragraph:

[0028] The opcode 230 uniquely encodes a value that specifies the test and skip instruction as described herein. The opcode 230 is read by the decode logic 152 (Figure 3). The first register reference Rd 232 preferably comprises three bits as shown and thus may reference any one of eight registers, for example, one of register R0-R7. The second register reference Rs 236 also preferably comprises three bits and thus also may reference one of eight registers such as registers R0-R7. Alternatively, the registers encoded in the Rd and Rs fields 232, 236 may include any other group of eight registers from registers ~~148~~ 140 (Figure 4). The P bit 234 generally dictates the type of addressing used for the instruction as explained below.

Please replace paragraph [0029] with the following amended paragraph:

[0029] The test and skip instruction ~~238~~ 228 preferably compares two operands to determine whether a condition is true. If the condition is true, then the instruction that follows the test and skip instruction ~~236~~ 228 is "skipped." Skipping the next instruction means that the subsequent instruction, which may have already been fetched by fetch logic 154, is not permitted to complete through the processor's pipeline. Skipping the subsequent instruction may occur by replacing the instruction with a "no operation" (NOP) instruction which is permitted to complete but, by its nature, generally does nothing.

Skipping the subsequent instruction may be performed in accordance with other ways as well, such as by flushing the subsequent instruction from the processor's pipeline.

Please replace paragraph [0030] with the following amended paragraph:

[0030] The operands compared in accordance with the test and skip instruction ~~238~~ 228 may be determined, at least in part, by the P bit 234. In accordance with at least some embodiments, one of the operands comprises the value stored in a register referenced by Rd 232. The second operand may be determined by the P bit 234. If the P bit 234 is set to a first value (e.g., "0"), the second operand preferably is the value stored in a register referenced by Rs 236. Accordingly, in this addressing mode (P bit set to a value of "0"), the contents of two registers are compared to each other. If, however, the P bit is set to a second value (e.g., "1"), the register referenced by Rs generally contains a pointer to a memory address. In this latter mode, the contents of a register are compared to a memory value pointed by another register. More specifically and in accordance with a preferred embodiment, the pointer to the memory location may be calculated by adding the contents of the register specified in the register reference 236 (Rs) to an implicit register (e.g., register R8) that stores an offset. The R8 register may be post-incremented by a suitable, predetermined value (e.g., 1) after the memory access to prepare the offset present in R8 for a subsequent access. Without limitation, this type of operation may be particularly useful when searching for a specific pattern in memory as in "code book searching." Accordingly, this test and skip instruction could be used within a software loop from a start address in memory.

Please replace paragraph [0031] with the following amended paragraph:

[0031] ~~The instruction operation is further illustrated in~~ Figure 6 further illustrates the test and skip instruction. As shown, the contents of a register ~~R2~~ Rd 232 is compared either to the contents of a register Rs 236 or to a memory location 242 in memory 244 to which Rs points. In either case, the operands are compared by applying a condition 240 (specified in the instruction 228). The result of the comparison dictates whether the subsequent instruction is skipped.

Please replace paragraph [0034] with the following amended paragraph:

[0034] Referring to Figures 4 and 5, in the event, the register referenced by Rd 232 is the top of stack register R7, the test and skip instruction ~~238~~ 228 causes the comparison to be made between a register (Rd) and the top of the stack. In this case, the execution of instruction ~~238~~ 228 may cause the stack pointer register R6 to be adjusted appropriately. That is, if the value at the top of the stack (e.g., the microstack 146) is used in the execution of the instruction, that top of stack value will be consumed. That being the case, the pointer to the top of the stack should be adjusted to point to the value that will subsequently represent the top of the stack. The adjustment to the stack pointer register R6 may be to decrement the stack pointer value by an appropriate amount (e.g., 1) depending on whether the stack management scheme is upwards or downwards based.

Please replace paragraph [0035] with the following amended paragraph:

[0035] The test and skip instruction described herein provides a variety of benefits, none of which should be used to narrow the scope of this disclosure. That being stated, such benefits may include optimizing searches through arrays for a specific value, minimum/maximum calculation, etc. The test and skip instruction generally also improves the overall execution time of Java code. Further, the test and skip instruction ~~238~~ 228 is generally relatively dense thereby permitting a Java application containing such instructions to require less memory storage space than otherwise would be needed.